Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process

Table of contents

1 Introduction

The deployment of small computing devices such as RFID tags, industrial controllers, sensor nodes and smart cards is becoming much more common. The shift from desktop computers to small devices brings a wide range of new security and privacy concerns. In many conventional cryptographic standards, the tradeoff between security, performance and resource requirements was optimized for desktop and server environments, and this makes them difficult or impossible to implement in resource-constrained devices. When they can be implemented, their performance may not be acceptable.

Lightweight cryptography is a subfield of cryptography that aims to provide solutions tailored for resource-constrained devices. There has been a significant amount of work done by the academic community related to lightweight cryptography; this includes efficient implementations of conventional cryptography standards, and the design and analysis of new lightweight primitives and protocols.

In 2013, NIST initiated a lightweight cryptography project to study the performance of the current NIST-approved cryptographic standards on constrained devices and to understand the need for dedicated lightweight cryptography standards, and if the need is identified, to design a transparent process for standardization. In July 2015, NIST held the first Lightweight Cryptography Workshop in Gaithersburg, MD, to get public feedback on the constraints and limitations of the target devices, and requirements and characteristics of real-world applications of lightweight cryptography. A second workshop was held in October 2016. In March 2017, NIST published NISTIR 8114 *Report on Lightweight Cryptography* and announced that it has decided to create a portfolio of lightweight algorithms through an open process. In April 2017, NIST published the draft whitepaper *Profiles for the Lightweight Cryptography Standardization Process* to solicit feedback on proposed functionalities for initial inclusion in the portfolio.

In this call for submissions document, the submission requirements and evaluation process for the lightweight cryptography standardization process are explained.

NIST requires that submission packages must be received by NIST by February 25, 2019 Submission packages that are received by NIST by January 4, 2019 will be reviewed for completeness by NIST; the submitters will be notified of any deficiencies within a month, allowing time for deficient packages to be amended by the submission deadline. After the submission deadline, NIST will publish all first-round submissions received, except that NIST may eliminate submissions that do not meet requirements stated in this call. No changes to packages will be permitted after the submission deadline, except at specified times during the evaluation phase.

Due to the specific requirements of the intellectual property statements as specified in Section 2.4, e-mail submissions **shall not** be accepted for these statements. The statements specified in Section 2.4 must be mailed to Dr. Kerry McKay, Information Technology Laboratory, Attention: Lightweight Cryptographic Algorithm Submissions, 100 Bureau Drive – Stop 8930, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930. The remainder of the submission package can either be mailed with the intellectual property statements, or sent as e-mail to: lightweight-crypto@nist.gov. This submission e-mail **shall** have subject line precisely "round 1 submission: NAME" where NAME is replaced by the name of the submission. For technical inquiries, send e-mail to lightweight-crypto@nist.gov. To facilitate the electronic distribution of submissions to all interested parties, copies of all written materials must also be submitted in electronic form in the PDF file format.

NIST welcomes both domestic and international submissions; however, in order to facilitate analysis and evaluation, it is required that the submission packages be in English. This requirement includes the cover sheet, algorithm specification and supporting documentation, source code comments, and intellectual property information.

"Complete and proper" submission packages will be posted at https://csrc.nist.gov/Projects/Lightweight-Cryptography for public review. To be considered as a "complete and proper" submission, packages **shall** satisfy the requirements specified in Section 2 and Section 3.

2 Requirements of Submission Packages

To be considered as a "complete" submission, packages **shall** contain the following:

- Cover sheet
- Algorithm specifications and supporting documentation
- Source code and test vectors
- Intellectual property statements / agreements / disclosures

These requirements are detailed below.

2.1 Cover Sheet

The cover sheet of a submission package **shall** contain the following information:

- Name of the submission.
- Name(s) of the submitter(s). Corresponding submitter's name, e-mail address, telephone, organization, and postal address.
- (optional) Backup point of contact (with telephone, postal address, and e-mail address).

2.2 Algorithm Specification and Supporting Documentation

A complete written specification of the algorithms **shall** be included, consisting of all necessary mathematical operations, equations, tables, and diagrams that are needed to implement the algorithms. The document **shall** also include a design rationale, and an explanation for all the important design decisions (with respect to targeted constrained devices) that have been made. The submitter **shall** explain the provenance of any constants or tables used in the algorithm.

Each submission package **shall** describe a single algorithm, or a collection of algorithms, that implements the authenticated encryption with associated data (AEAD) functionality, and optionally also implements the hashing functionality.

For algorithms that have tunable parameters, the submission document **shall** specify concrete values for these parameters. The submission may specify several parameter sets that allow the selection of a range of possible security/performance tradeoffs. The submitter **shall** provide an analysis of how the security and performance of the algorithms depend on these parameters.

The submission package **shall** include a statement of the expected security strength of each variant of the submission, along with a supporting rationale. The submission package **shall** include a statement that summarizes the known cryptanalytic attacks on the variants of the submission, and provide estimates of the complexity of these attacks.

The submission package **shall** include a statement that lists and describes the advantages and limitations of the cryptosystem in terms of security, performance, and implementation costs (e.g., estimates for required RAM, ROM, or gate equivalents).

The submission of algorithms that are not well-understood is discouraged. Submissions are expected to have third-party analysis of the design, or leverage existing standards or heavily-analyzed components as part of the design. The submitter **shall** provide a list of references to any published materials describing or analyzing the security of the submitted algorithm or cryptosystem.

2.3 Source Code and Test Vectors

A reference implementation **shall** be provided with the submission package. The goal of the reference implementation is to promote the understanding of how the submitted algorithm may be implemented and also to allow the verification of the optimized implementations. It **shall not** contain any optimizations that will make it more difficult to understand the details of the algorithm. The source code **shall** be accompanied by a set of test vectors that will be generated by the submitter. Information on how the source code and the test vectors should be compiled together to form the *source code package* can be found in Section 3.5.

2.4 Intellectual Property Statements / Agreements / Disclosures

Each submitted algorithm, together with each submitted reference implementation and optimized implementation (if any), must be made freely available for public review and evaluation purposes worldwide during the standardization period.

Given the nature and use of cryptographic algorithms, NIST's goals include identifying technically robust algorithms and facilitating their widespread adoption. NIST does not object in principle to algorithms or implementations which may require the use of a patent claim, where technical reasons justify this approach, but will consider any factors which could hinder adoption in the evaluation process.

NIST has observed that royalty-free availability of cryptosystems and implementations has facilitated adoption of cryptographic standards in the past. For that reason, NIST believes it is critical that this process leads to cryptographic standards that can be freely implemented in security technologies and products. As part of its evaluation of a cryptographic algorithm for standardization, NIST will consider assurances made in the statements by the submitter(s) and any patent owner(s), with a strong preference for submissions as to which there are commitments to license, without compensation, under reasonable terms and conditions that are demonstrably free of unfair discrimination.

The following signed statements will be required for a submission to be considered 'complete':

1) Statement by each submitter,
2) Statement by patent (and patent application) owner(s) (if applicable), and
3) Statement by reference/optimized implementations' owner(s).

Note that for the last two statements, separate statements must be completed if multiple individuals are involved.

2.4.1 Statement by Each Submitter

I, _____ (print submitter's full name), of ____ (print full postal address), do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as ______ (print name of cryptosystem), is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that (check at least one of the following):

☐ *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as ____ (print name of cryptosystem);*

☐ *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as ____ (print name of cryptosystem), may be covered by the following U.S. and/or foreign patents: _____ (describe and enumerate or state "none" if applicable)_____ ;*

☐ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: _____ (describe and enumerate or state "none" if applicable) ______.*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment I do hereby agree to provide the statements required by Sections 2.4.2 and 2.4.3, below, for any patent or patent application identified to cover the practice *of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.*

I acknowledge that, during the lightweight crypto evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.4.1, 2.4.2 and 2.4.3, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed:

Title:

Date:

Place:

2.4.2 Statement by Patent (and Patent Application) Owner(s)

If there are any patents (or patent applications) identified by the submitter, including those held by the submitter, the following statement must be signed by each and every owner, or each owner's authorized representative, of each patent and patent application identified.

I, _____ (print full name), of _____(print full postal address), am the owner or authorized representative of the owner (print full name, if different than the signer) of the following patent(s) and/or patent application(s): ______ (enumerate), and do hereby commit and agree to grant to any interested party on a worldwide basis, if the cryptosystem known as _____(print name of cryptosystem) is selected for standardization, in consideration of its evaluation and selection by NIST, a non-exclusive license for the purpose of implementing the standard (check one):

☐ *without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination,* ***OR***

☐ *under reasonable terms and conditions that are demonstrably free of any unfair discrimination.*

I further do hereby commit and agree to license such party on the same basis with respect to any other patent application or patent hereafter granted to me, or owned or controlled by me, that is or may be necessary for the purpose of implementing the standard.

I further do hereby commit and agree that I will include, in any documents transferring ownership of each patent and patent application, provisions to ensure that the commitments and assurances made by me are binding on the transferee and any future transferee.

I further do hereby commit and agree that these commitments and assurances are intended by me to be binding on successors-in-interest of each patent and patent application, regardless of whether such provisions are included in the relevant transfer documents.

I further do hereby grant to the U.S. Government, during the public review and the evaluation process, and during the lifetime of the standard, a nonexclusive, nontransferable, irrevocable, paid-up worldwide license solely for the purpose of modifying my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability) for incorporation into the standard.

Signed:

Title:

Date:

Place:

2.4.3 Statement by Reference/Optimized/Additional Implementations' Owner(s)

The following must also be included:

I, _____ (print full name), _______(print full postal address), am the owner or authorized representative of the owner (print full name, if different than the signer) of the submitted reference implementation, optimized and additional implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the lightweight cryptography public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed:

Title:

Date:

Place:

3 Minimum Acceptability Requirements

To be considered as a "proper" submission, packages **shall** satisfy the requirements stated in this section. The following requirements have some similarities with the call of Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) (https://competitions.cr.yp.to/caesar-call.html) and eBACS: ECRYPT Benchmarking of Cryptographic Systems (https://bench.cr.yp.to/), for which the unified SUPERCOP benchmarking suite was developed. This has been done to facilitate the submission of algorithms, and the benchmarks of algorithm performance.

3.1 AEAD Requirements

An *authenticated encryption with associated data (AEAD) algorithm* is a function with four byte-string inputs and one byte-string output. The four inputs are a variable-length *plaintext*, variable-length *associated data*, a fixed-length *nonce*, and a fixed-length *key*. The output is a variable-length *ciphertext*. Authenticated decryption, also known as decryption-verification, **shall** be supported: it **shall** be possible to recover the plaintext from a valid ciphertext (i.e., a ciphertext that corresponds to the plaintext for a given associated data, nonce, and key), given associated data, nonce and key. Plaintext **shall not** be returned by the decryption-verification process if the ciphertext is invalid.

From a security point of view, an AEAD algorithm should ensure both the confidentiality of the plaintexts (under adaptive chosen-plaintext attacks) and the integrity of the ciphertexts (under adaptive forgery attempts). AEAD algorithms are expected to maintain security as long as the nonce is unique (not repeated under the same key). Any security loss when the nonce is not unique **shall** be documented, and algorithms that do not lose all security with repeated nonces may advertise this as a feature.

The submitters are allowed to submit a family of AEAD algorithms, where members of the family may vary in external parameters (e.g., key length, nonce length), or in internal parameters (e.g., number of rounds, or state size). The family **shall** include at most 10 members. The following requirements apply to all members of the family.

An AEAD algorithm **shall** not specify key lengths that are smaller than 128 bits. Cryptanalytic attacks on the AEAD algorithm **shall** require at least 2^{112} computations on a classical computer in a single-key setting. If a key size larger than 128 bits is supported, it is recommended that at least one recommended parameter set has a key size of 256 bits, and that its resistance against cryptanalytical attacks is at least 2^{224} computations on a classical computer in a single-key setting.

AEAD algorithms **shall** accept all byte-string inputs that satisfy the input length requirements. Submissions **shall** include justification for any length limits.

The family **shall** include one *primary* member that has a key length of at least 128 bits, a nonce length of at least 96 bits, and a tag length of at least 64 bits. The limits on the input sizes (plaintext, associated data, and the amount of data that can be processed under one key) for this member **shall not** be smaller than 2^{50}-1 bytes.

3.2 Hash Function Requirements

A *hash function* is a function with one byte-string input and one byte-string output. The input is a variable-length *message*. The output is a fixed-length *hash value*.

It should be computationally infeasible to find a collision or a (second) preimage for this hash function. The hash function should also be resistant against length extension attacks. For example, if part of the message is a secret key that is unknown to the attacker, it should be infeasible for this attacker to construct a hash value corresponding to a different message that contains the same secret key. In several practical applications, hash functions may need to satisfy other security properties as well, such as retaining some level of security when the output is truncated. Hash function submissions should describe any additional security properties that are provided.

The submitters are allowed to submit a family of hash functions, where members of the family may vary in external parameters (e.g., maximum message length, output size), or in internal parameters (e.g., number of rounds, or state size). The family **shall** include at most 10 members. The following requirements apply to all members of the family.

Cryptanalytic attacks on the hash function **shall** require at least 2^{112} computations on a classical computer. The hash function **shall not** specify output sizes that are smaller than 256 bits.

Hash functions **shall** accept all byte-string inputs that meet the specified maximum length of messages. Submissions **shall** include justification for any length limits.

The family **shall** include one *primary* member that has an output size of at least 256 bits. The limit on the message size for this member **shall** not be smaller than 2^{50}-1 bytes.

3.3 Additional Requirements for Submissions with AEAD and Hashing

This section provides additional requirements on the submissions that provide both AEAD and hashing functionality.

Submissions **shall** state which design components the AEAD and hashing algorithms have in common, and explain how these common components lead to a reduced implementation cost.

Submissions **shall** specify list of pairs of AEAD and hash function family members to be evaluated jointly. This list is permitted to be as short as one recommendation. The primary member of the AEAD family and the primary member of the hash function family **shall** be paired together. This list **shall** not be longer than ten recommendations.

3.4 Design Requirements

Submitted AEAD algorithms and optional hash function algorithms should perform significantly better in constrained environments (hardware and embedded software platforms) compared to current NIST standards. They should be optimized to be efficient for short messages (e.g., as short as 8 bytes). Compact hardware implementations and embedded software implementations with low RAM and ROM usage should be possible. The performance on ASIC and FPGA should consider a wide range of standard cell libraries. The algorithms should be flexible to support various implementation strategies (low energy, low power, low latency). The performance on microcontrollers should consider a wide range of 8-bit, 16-bit and 32-bit microcontroller architectures. For algorithms that have a key, the preprocessing of a key (in terms of computation time and memory footprint) should be efficient.

The implementations of the AEAD algorithms and the optional hash function algorithms should lend themselves to countermeasures against various side-channel attacks, including timing attacks, simple and differential power analysis (SPA/DPA), and simple and differential electromagnetic analysis (SEMA/DEMA).

Designs may make tradeoffs between various performance requirements. A submission is allowed to prioritize certain performance requirements over others. To satisfy the stringent limitations of some constrained environments, it may not be possible to meet all performance requirements stated in the previous paragraph. The submission document should, however, explain the bottlenecks that were identified and the tradeoffs that were made.

3.5 Implementation Requirements[1]

Each submission **shall** be accompanied by a portable reference software implementation, in C, to support public understanding of the algorithms, cryptanalysis, verification of subsequent implementations, etc. An implementation **shall** be provided for all members of the family, and **shall** compute exactly the functions specified in the submission. This reference implementation is expected to be easy to understand, and should not include code that is solely intended to optimize performance on certain platforms. For example, the reference implementation **shall not** contain compiler intrinsics, platform-specific headers, or compiler-specific features. The submission may also include optimized implementations that use the same API, or additional implementations that highlight specific implementation features of the algorithms. There are no restrictions on the API for additional implementations.

The correctness of the reference implementation **shall** be verified on the NIST test vector verification platform. This platform is an Intel x64-based system, running Ubuntu 16.04 (64-bit) and the reference implementations **shall** be compiled with GCC 5.4.0 using the compiler flags:

```
-std=c99 -Wall -Wextra -Wshadow -fsanitize=address,undefined -O2
```

The following sections describe specific implementation requirements for AEAD and hash functions.

3.5.1 AEAD

A minimal reference implementation of a variant of an AEAD algorithm consists of two files: `api.h` and `encrypt.c`. As an example, MyAEAD algorithm with 256-bit keys would consist of the files: `crypto_aead/myaead256v1/ref/api.h` and `crypto_aead/myaead256v1/ref/encrypt.c`. There are three levels of directory names:

- The first-level directory name `crypto_aead` is the same for all AEAD algorithms.
- The second-level directory name is a lowercase version of the name of the algorithm, including the version number and a family member identifier (if multiple family members in submission). A reference implementation covering multiple family members must have a second-level directory for each member. Dashes, dots, slashes, and other punctuation marks are omitted; the directory name consists solely of digits (`0123456789`) and lowercase ASCII letters (`abcdefghijklmnopqrstuvwxyz`).
- The third-level directory name is `ref` for the reference implementation. Other implementations of the same AEAD algorithm (with the same parameter set) use other third-level directory names. Third-level directory names starting with `add_` may be used

[1] Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

only for the additional implementations that highlight specific implementation features of the algorithms. These include software implementations that do not satisfy the API, as well as any hardware implementations that may be included in the submission.

- For each of the implementations that satisfy the API, the `genkat_aead.c` file[2] **shall** be used to generate the test vector output file. The submitters **shall** verify that the test vector output files are identical for all implementations in the second-level directory, and **shall** provide the test vector output file in the `crypto_aead/myaead256v1/` directory for the variant of the MyAEAD algorithm with 256-bit keys in the aforementioned example.

The file api.h consists of five definitions. For example:

```
#define CRYPTO_KEYBYTES 32
#define CRYPTO_NSECBYTES 0
#define CRYPTO_NPUBBYTES 12
#define CRYPTO_ABYTES 16
#define CRYPTO_NOOVERLAP 1
```

This indicates that for this variant of the MyAEAD algorithm, the key is 32 bytes, the nonce is 12 bytes, and that the ciphertext is *at most* 16 bytes longer than the plaintext. (A typical AEAD algorithm has a constant gap between plaintext length and ciphertext length, but the requirement here is to have a constant *limit* on the gap.) The definition `CRYPTO_NSECBYTES` **shall** always be set to zero.

The last definition `CRYPTO_NOOVERLAP` is an optional definition in SUPERCOP API and indicates whether the implementation can handle overlapping input and output buffers. To ensure compatibility with the SUPERCOP API, `api.h` file **shall** contain "`#define CRYPTO_NOOVERLAP 1`". Regardless of whether this flag is needed in the SUPERCOP framework, it clarifies how the API is intended to be used; the implementation is not expected to handle overlapping input and output buffers. (Note that if `CRYPTO_NOOVERLAP` is not defined, the SUPERCOP framework assumes that inputs and outputs can overlap, and returns an error if this behavior is not supported.)

The file `encrypt.c` has the following structure:

```
#include "crypto_aead.h"

int crypto_aead_encrypt(
  unsigned char *c,unsigned long long *clen,
  const unsigned char *m,unsigned long long mlen,
  const unsigned char *ad,unsigned long long adlen,
  const unsigned char *nsec,
  const unsigned char *npub,
  const unsigned char *k
)
{
  ...
  ... the code for the cipher implementation goes here,
```

[2] Available at: https://csrc.nist.gov/Projects/Lightweight-Cryptography

```
  ... generating a ciphertext c[0],c[1],...,c[*clen-1]
  ... from a plaintext m[0],m[1],...,m[mlen-1]
  ... and associated data ad[0],ad[1],...,ad[adlen-1]
  ... and nonce npub[0],npub[1],...
  ... and secret key k[0],k[1],...
  ... the implementation shall not use nsec
  ...
  return 0;
}

int crypto_aead_decrypt(
  unsigned char *m,unsigned long long *mlen,
  unsigned char *nsec,
  const unsigned char *c,unsigned long long clen,
  const unsigned char *ad,unsigned long long adlen,
  const unsigned char *npub,
  const unsigned char *k
)
{
  ...
  ... the code for the AEAD implementation goes here,
  ... generating a plaintext m[0],m[1],...,m[*mlen-1]
  ... and secret message number nsec[0],nsec[1],...
  ... from a ciphertext c[0],c[1],...,c[clen-1]
  ... and associated data ad[0],ad[1],...,ad[adlen-1]
  ... and nonce number npub[0],npub[1],...
  ... and secret key k[0],k[1],...
  ...
  return 0;
}
```

The outputs of `crypto_aead_encrypt` and `crypto_aead_decrypt` **shall** be determined entirely by the inputs listed above (except that the parameter `nsec` is kept for compatibility with SUPERCOP and will not be used) and **shall** not be affected by any randomness or other hidden inputs.

The `crypto_aead_decrypt` function **shall** return -1 if the ciphertext is not valid. The `crypto_aead_encrypt` and `crypto_aead_decrypt` functions may return other negative numbers to indicate other failures (e.g., memory-allocation failures).

The file `crypto_aead.h` contains the declarations of the API functions, and should not be modified in any way. It should not be included in the reference implementation.

A reference implementation can use names other than `encrypt.c`. It can split its code across several files `*.c` defining various auxiliary functions; the files will be automatically compiled together.

3.5.2 Hash Function

A minimal reference implementation of a hash function consists of two files: api.h and hash.c. As an example, `MyHash` hash function with 256-bit output would consists of the files: `crypto_hash/myhash256v1/ref/api.h` and `crypto_hash/myhash256v1/ref/hash.c`. There are three levels of directory names:

- The first-level directory name `crypto_hash` is the same for all hash functions.
- The second-level directory name is a lowercase version of the name of the algorithm, including the version number and a family member identifier (if multiple family members in submission). A reference implementation covering multiple family members must have a second-level directory for each member. Dashes, dots, slashes, and other punctuation marks are omitted; the directory name consists solely of digits (`0123456789`) and lowercase ASCII letters (`abcdefghijklmnopqrstuvwxyz`).
- The third-level directory name is `ref` for the reference implementation. Other implementations of the same hash function (with the same parameter set) use other third-level directory names. Third-level directory names starting with `add_` may be used only for the additional implementations that highlight specific implementation features of the algorithms. These include software implementations that do not satisfy the API, as well as any hardware implementations that may be included in the submission.
- For each of the implementations that satisfy the API, the `genkat_hash.c` file[3] **shall** be used to generate the test vector output file. The submitters **shall** verify that the test vector output files are identical for all implementations in the second-level directory, and **shall** provide the test vector output file in the `crypto_hash/myhash256v1/` directory for the 256-bit output of the MyHash hash function in the aforementioned example.

The file `api.h` contains one definition. For example:

```
#define CRYPTO_BYTES 32
```

This indicates that for this variant of the MyHash algorithm, the output size is 32 bytes.

The file `hash.c` has the following structure:

```
#include "crypto_hash.h"

int crypto_hash(
  unsigned char *out,
  const unsigned char *in,
  unsigned long long inlen
)
{
  ...
  ... the code for the hash function implementation goes here
```

[3] Available at: https://csrc.nist.gov/Projects/Lightweight-Cryptography

```
    ... generating a hash value out[0],out[1],...,out[CRYPTO_BYTES-1]
    ... from a message in[0],in[1],...,in[in-1]

    ...
    return 0;
  }
```

To ensure compatibility with the SUPERCOP, the implementation of `crypto_hash` **shall** handle overlapping input and output buffers.

The output of `crypto_hash` **shall** be determined entirely by the message input and **shall** not be affected by any randomness or other hidden inputs.

The `crypto_hash` function may return a negative number to indicate other failure (e.g., memory-allocation failures).

The file `crypto_hash.h` contains the declaration of the API function, and should not be modified in any way. It should not be included in the reference implementation.

A reference implementation can use names other than `hash.c`. It can split its code across several files `*.c` defining various auxiliary functions; the files will be automatically compiled together.

Finally, all implementations **shall** be packaged into a tarball, such as mysubmissionv1.tar.gz for a reference implementation of MySubmission v1, including all members of the MyAEAD v1 family of AEAD algorithms. If the submission specifies a hash function, it will also include the members of the MyHash v1 family of hash functions. This tarball **shall** be included in the submission package.

4 Evaluation Criteria

4.1 Minimum Acceptability of the Submission

The evaluation will verify whether the submission meets the minimum acceptability requirements, as described in Section 3 of this notice. This will include a security evaluation of the algorithms against known attacks (e.g., differential cryptanalysis) that may violate the minimum submission requirements.

4.2 Side Channel and Fault Attack Resistance

Side channel resistance is the ability for an implementation to reduce the information gained by measurable phenomena about the inner workings of a cryptographic computation (such as timing, power, electromagnetic field, ciphertext length). While implementations will not be required to provide side channel resistance, the ability to provide it easily and at low cost is highly desired. Side channel resistance may be necessary in some applications.

A fault attack alters the normal functioning of a physical electronic device (e.g., by changing the supply voltage), such that it causes an error in the computation that can be leveraged to perform an attack (e.g., key recovery). Resistance to fault attacks may also be a desired feature in applications. While implementations will not be required to protect against fault attacks, the ability to provide it easily and at low cost will be taken into consideration.

4.3 Cost

Submissions will be evaluated in terms of various cost metrics (e.g., area, memory, energy consumption), as appropriate.

4.4 Performance

Submissions will be evaluated in terms of various performance metrics (e.g., latency, throughput, power consumption), as appropriate.

4.5 Third-party Analysis

Submissions that have significant third-party analysis or leverage components of existing standards will be favored for selection.

4.6 Suitability for Hardware and Software Implementations

An algorithm may be well-suited for both hardware and software, or it may be specifically tailored for performance in either one. Submissions that perform well in both will likely be given greater consideration; however, a submission that excels in highly-constrained hardware may also be granted greater consideration for selection.

5 Evaluation Process

NIST will form an internal selection panel composed of NIST researchers to analyze the submissions. All of NIST's analysis results will be made publicly available.

Although NIST will be performing its own analyses of the submitted algorithms, NIST strongly encourages public evaluation and publication of the results. NIST will take into account its own analysis, as well as the public comments that are received in response to the posting of the "complete and proper" submissions, to make its decisions.

Following the close of the call for submission packages, NIST will review the received packages to determine which are "complete and proper," as described in Sections 2 and 3 of this notice. NIST will post all "complete and proper" submissions at https://csrc.nist.gov/Projects/Lightweight-Cryptography for public review.

The initial phase of evaluation will consist of approximately twelve months of public review of the submitted algorithms. During this initial review period, NIST intends to evaluate the submitted algorithms as outlined in Section 4. Depending on the number of submissions, NIST may eliminate algorithms from consideration early in the first evaluation phase in order to focus analysis on the strongest submissions. A workshop will be held ten to eleven months after the submission deadline to discuss analysis of first round candidates. NIST will review the public evaluations of the submitted algorithms' cryptographic strengths and weaknesses, implementation costs, and implementation performance and will use these to narrow the candidate pool for more careful study and analysis. The purpose of this selection process is to identify candidates that are suitable for standardization in the near future. Algorithms that are not included in the narrowed pool may still be considered for standardization at a later date, unless they are explicitly removed from consideration by NIST or the submitter.

Because of limited resources, and also to avoid moving evaluation targets (i.e., modifying the submitted algorithms undergoing public review), NIST will not accept substantive modifications to the submitted algorithms during this initial phase of evaluation.

For informational and planning purposes, near the end of the initial public evaluation process, NIST intends to hold another lightweight cryptography standardization conference. Its purpose will be to publicly discuss the submitted algorithms, and to provide NIST with information for narrowing the field of algorithms for continued evaluation.

NIST plans to narrow the field of algorithms for further study, based upon its own analysis, public comments, and all other available information. It is envisioned that this narrowing will be done primarily on security, cost, performance, and intellectual property considerations. NIST will issue a report describing its findings.

During the course of the initial evaluations, it is conceivable that some small deficiencies may be identified in even some of the most promising submissions. Therefore, for the second round of evaluations, small modifications to the submitted algorithms will be permitted for either security or efficiency purposes. Submitters may submit minor changes (no substantial redesigns), along with a supporting justification that must be received by NIST prior to the beginning of the second evaluation period. (Submitters will be notified by NIST of the exact deadline.) NIST will determine whether the proposed modification would significantly affect the design of the algorithm, requiring a major re-evaluation; if such is the case, the modification will not be accepted. If modifications are submitted, new reference and optimized implementations and written descriptions must also be provided by the announced deadline. This will allow a thorough public review of the modified algorithms during the entire course of the second evaluation phase.

Note that all proposed changes **shall** be conveyed by the submitter; no proposed changes (to the algorithm or implementations) will be accepted from a third party.

The second round of evaluation will consist of approximately nine to twelve months of public review, with a focus on a narrowed pool of candidate algorithms. During the public review, NIST will similarly evaluate these algorithms. After the end of the public review period, NIST intends to hold another lightweight cryptography standardization conference.

Following the third lightweight cryptography standardization conference, NIST will prepare a summary report, which may select algorithm(s) for possible standardization. Any selected algorithm(s) for standardization will be incorporated into draft standards, which will be made available for public comment.

Specific parameters will be chosen during the standardization process following the final evaluation phase. Specific parameter sets may permit NIST to select a different performance/security tradeoff than originally specified by the submitter, in light of discovered attacks or other analysis. NIST will consult with the submitter of the algorithm, as well as the cryptographic community, if it plans to select that algorithm for development as a NIST standard with a different parameter set than originally specified by the submitter.

When evaluating algorithms, NIST will make every effort to obtain public input and will encourage the review of the submitted algorithms by outside organizations. NIST encourages the reviewers to demonstrate their findings and attacks both on the versions with parameters that achieve full security levels, as well as with practical attacks on the provided parameter sets with lower security levels. The final decision as to which (if any) algorithm(s) will be selected for standardization is the responsibility of NIST.

It should be noted that this schedule for the evaluation process is somewhat tentative, depending upon the type, quantity, and quality of the submissions. Specific conference dates and public comment periods will be announced at appropriate times in the future. NIST estimates that some algorithms could be selected for standardization after two to four years. However, due to developments in the field, this timeline could change.